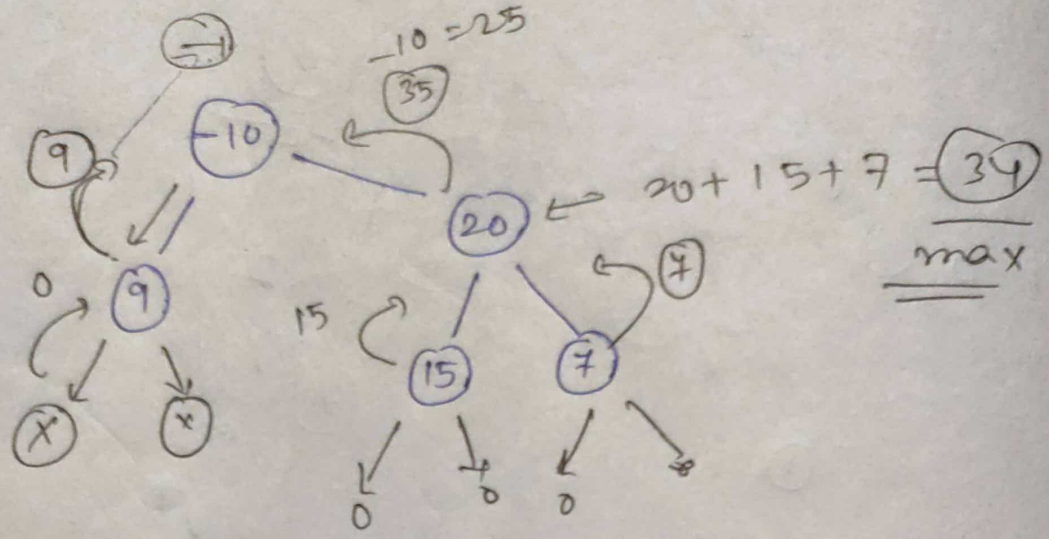
Maximum Path Sum:-



Pseudo Code:-

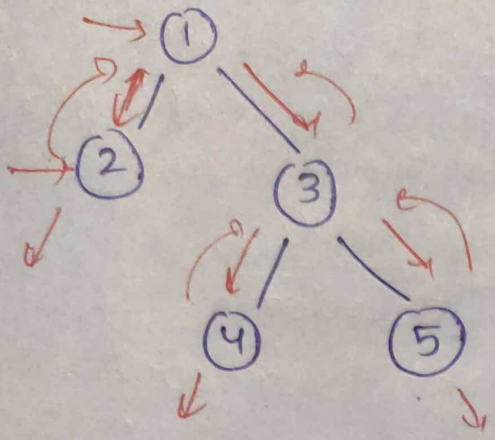
```
int maxpath(node, maxi){
    if(node == null)
        return 0;
    leftsum = maxpath(node->left);
    rightsum = maxpath(node->right);
maxi = max(maxi, leftsum+rightsum+node->val);
    return (node->val)+max(leftsum, rightsum)
```

Code:-

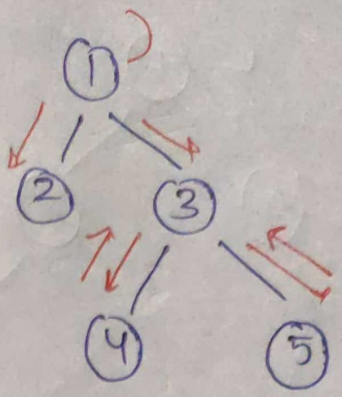
```
int maxPathSum(TreeNode *root){
        int maxi = INT_MIN;
        maxPathDown(root, maxi);
        return maxi;
    }
int maxPathDown(TreeNode *root, int &maxi){
    if(node == NULL) return 0;
    int left = max(0, maxPathDown(node->left,
int right = max(0, " (node->right, maxi);
maxi = max(maxi, left+right+node->val);
return max(left,right)+node->val;
}
```

Check for ~~Balanced~~ Identical Tree :-

Preorder → Root, left, right.



Tree → 1

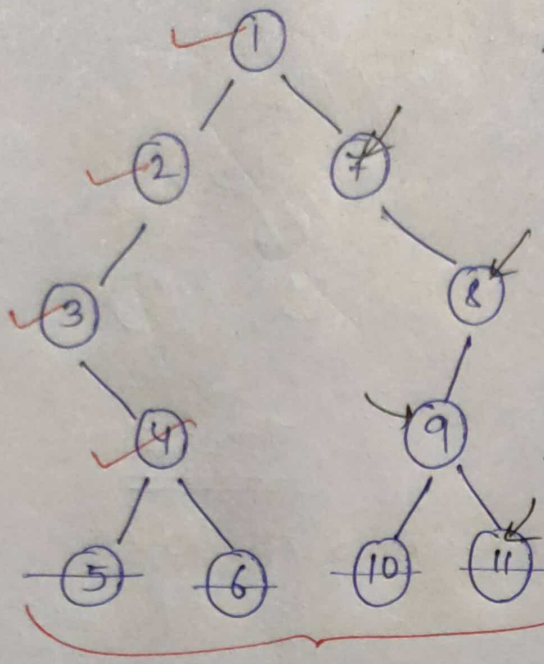


Tree-2

Use any traversal technique to check for Balanced Trees.

```
bool isSameTree(TreeNode *p, TreeNode *q) {
    if(p==NULL || q==NULL) {
        return (p==q);
    }
    return (p->val==q->val)
        && isSameTree(p->left, q->left):
        && isSameTree(p->right, q->right);
}
```

Boundary Traversal of a Binary Tree (Anticlockwise)

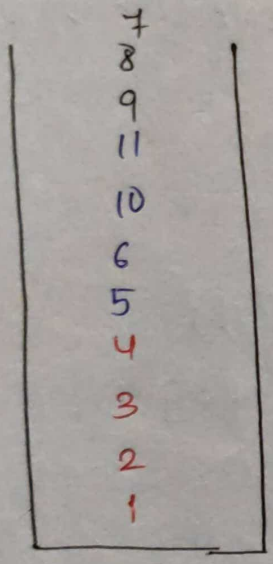


→ Left Boundary excluding leaf node.
↳ Traverse left subtree

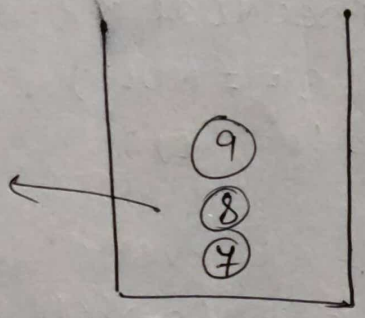
→ Leaf Nodes.
Inorder

→ Right Boundary (Reverse) excluding leaf node.

Inorder



ds



st.

```
vector<int> printBoundary(Node *root){
    vector<int> res;
    if(!root) return res;
    if(!isLeaf(root)) res.push_back(root->data);
    addLeftBoundary(root,res);
    // add leaf nodes.
    addLeaves(root,res);
    addRightBoundary(root,res);
    return res;
}
```

Code :-

```
bool isleaf(Node *root) {
    return !root->left && !root->right;
}

void addleftBoundary(Node *root, vector<int> &res){
    Node *cur = root->left;
    while(cur) {
        if(!isleaf(cur)) res.push_back(cur->data);
        if(cur->left) cur = cur->left;
        else cur = cur->right;
    }
}

void addRightBoundary(Node *root, vector<int> &res){
    Node *cur = root->right;
    vector<int> temp;
    while(cur) {
        if(!isleaf(cur)) temp.push_back(cur->data);
        if(cur->right) cur = cur->right;
        else cur = cur->left;
    }
    for(int i = temp.size()-1; i>=0; i--){
        res.push_back(temp[i]);
    }
}

void addleaves(Node *root, vector<int> &res) {
    if(isleaf(root)){
        res.push_back(root->data);
        return;
    }
    if(root->left) addleaves(root->left, res);
    if(root->right) addleaves(root->right, res);
}
```